Evolvable Hardware for Spacecraft Autonomy

Alex Fukunaga, Ken Hayworth, Adrian Stoica
Jet Propulsion Laboratory
4800 Oak Glow 1)[.
Pasadena, CA 91109
626-3066 I 57
alex.fukunaga@jpl.nasa.gov

Abstract—*Evolvable hardware* is a recently proposed technology ill which reconfigurable hardware under the control of an evolutionary (genetic) algorithm can automatically self-reconfigure into configurations with the desired behavior. This w ould not only enable the on-demand gene.ration 01 new functional ity when needed, but (his could provide increased fault-tole rance, as the hardware would be able to cope with faults by reassigning function cells 10 take over the faulty ones. This paper describes ongoing" work at JPL, focusing 011 applications to onboard image processing.

TABLE OF CONTENTS

1. INTRODUCTION

Spacecraft autonomy plays a key role in future NASA missions. An intelligent, autonomous spacecraft must be able to cope with problems for which solutions were not specified on ground, and adapt itself to ncw or changing environments. Ultimately, all adaptations originate in the on-board electronics that control such changes. Thus, it is important 1 0 address on-board electronics with the capability to *evolve* - modify itself to provide increased efficiency of the systems it controls. While for analog circuitry it is clear why circuit modifications are needed for modifying/adapting their performance, the adaptation 0 1 computer-related functions could, in principle, be controlled by software running (m general purpose flight processors." However, hardware with a design optimized for certain functionality would provide much higher processing power. Such increased performance could be achieved using *reconfigurable hardware,* for example built with field-programmable gate arrays (FPGA). At present, the architectures downloaded for I PGA configuration are designed by humans.

Evolvable hardware is a new technology in which reconfigurable hardware under the control of an evolutionary (genetic) algorithm can automatically self-reconfigure into configurations with the desired behavior. This would not only enable the on-demand generation of new functionality when needed, but ibis could provide increased fault-toler ance, as the hardware would be able to cope with faults by reassigning function cells to take over the faulty ones.

Figure. A schematic representation of evolvable hardware which, through a succession of changes of its cell functions and connectivity patterns, reaches a structure producing a desired response (here, a sine wave).

Extrinsic EHW refers to evolution in a software simulation using models 0 1 the hardware behavior, downloading" the configuration of the best evolved architecture [() programmable hardware. in *intrinsic* evolvable hardware (to which most of the following" discussion applies) configuration bits are iteratively downloaded to hardware, evaluating a degree O r adaptation/fitness by observing the behavior of the real hardware.

Hardware evolution is performed through a succession of changes o f elementary cell functions and cell interconnectivity pattern, thus obtaining increasingly more fit configurations until a target functionality is reached. As it is the case in nature, evolution results in individuals that are increasingly more adapted 10 their environments, and can change themselves 10 match changes in environments and modifications of their own goals. Unlike in nature, evolution in silicon has the advantage that L'mild be extremely rapid, with millions of generations or "living" cite.Llils evaluated in only a few seconds.

Hardware evolution can be seen as an on-chip search for the circuit/configuration whose behavior is closest to the

required one (e.g. gives best performance/adaptation to the environment). The suitability for a parallel hardware implementation of evolvable hardware, with multiple "islands" of concurrently evolving circuits on the same chip, or in a multi-chip or stacked configuration is very attractive.

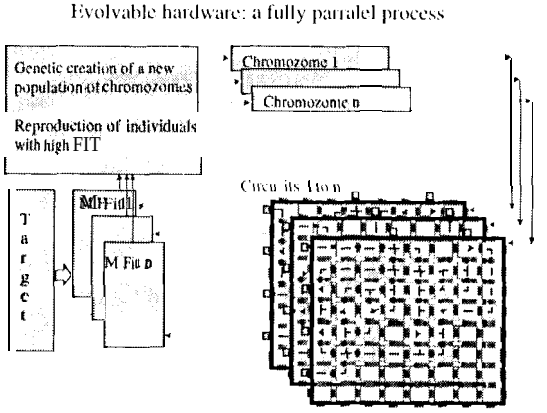

Figure. Parallel implementation for evolvable hardware

The granularity of hardware building blocks for those attempting intrinsic evolvable hardware is currently influenced by the availability of certain programmable devices. The paper presents results of simulated evolution at transistor level and discusses on the role of the level of granularity and evolvability.

In this paper, we will briefly review the state of the art in the emerging field of evolvable hardware, and identify the critical issues which must be addressed in order for the application of this new technology to be feasible to NASA missions. We will then describe early results of ongoing research at JPL, on evolvable hardware (these are briefly described below).

2. Related Work

This section briefly reviews the related work in the field of evolvable hardware.

Thompson [Tho96a,Tho96b] used a genetic algorithm [Hol75] to evolve a Xilinx XC6216 FPGA configuration to perform tone discrimination between 1 kHz and 10kHz waves (by searching the space of possible FPGA configuration strings).

Higuchi et al [HIM+97] developed a custom FPGA in which function blocks perform arithmetic functions, in addition to lower-level logical functions (standard FPGA function blocks perform only logical functions). They have shown in simulation that this custom FPGA can be used to evolve image compression" algorithms and equalizers.

Koza et al [KBAK96] have applied genetic programming to evolve a wide range of analog circuits. Their system generates SPICE simulatable netlists. However, although their evolved solutions can be simulated, they cannot be implemented in practice (e.g., ideal components are assumed).

Hemmi et al [HMS97] - have developed the AdAM system, which uses genetic programming to generate VHDL descriptions of digital hardware. This effort is closest in spirit to our own although each. However, Hemmi et al have (rely reported results using a toy problem (the simulated ant trail-following (navigation) problem, which was introduced in [JCC92] and is a standard testbed problem in the evolutionary computing community). Furthermore, in comparison with the s-expression representation we use, VHDL is a significantly lower level representation language (as we note below, we believe that the use of a high-level representation is crucial for scaling up evolvable hardware to complex, high-level tasks such as image processing).

While the previous work summarized above has demonstrate the potential of self-reconfiguring hardware, they present several major open problems which needs to be addressed in order for self-reconfiguring hardware to become a viable technology for JPL/NASA. These include:

- *scaling to complex/higher-level problems:* Can evolvable hardware be applied to complex tasks which are of interest to JPL/NASA, such as image processing? With the exception of the work at ETL [Hir96,Hir97], the existing work on evolvable hardware focuses on low-level, analog tasks [KBAK96] and "toy problems" such as tone discrimination [Tho96a,Tho96b] and simplified navigation tasks [HHS97].
- *generality:* is it possible to develop a general-purpose evolvable hardware system which can be applied easily to a wide range of tasks? This would be preferable to requiring that a spacecraft/robot have on-board a different system for every application which requires adaptation/self-configuration.
- *commodity technology for digital reconfigurable hardware:* is it possible to use industry-developed hardware platforms (i.e., Field Programmable Gate Arrays) as the underlying reconfigurable hardware platform? This would let us leverage the significant industry R&D investment in FPGAs and provide us with a stable, high-performance, reconfigurable hardware platform. While previous researchers have used commodity FPGAs [Tho96a,HHS97], these have been restricted to toy problems, and the question

of whether standard FPGAs can be used for complex tasks has remained an open problem.

3. EVOLUTION 0}' HIGH-LEVEL SIGNAL PROCESSING ALGORITHMS ON FPGAS

in this section, we describe ongoing work which seeks to address all three of the above issues: Wc have developed an approach in which algorithms (represente d as 1 .isp functions) are evolved using genetic programming [Koz92]. These algorithms are then mapped onto a commodity reconfigurable hardware platform, e.g., a field-programmable gale array (FPGA) using VI ,S1 CAD bigl~-level synthesis tools. Preliminary results with ibis approach in the domain of lossless image compression are reported.

Most existing work on digital evolvable hardware is fundamentally limited in scalability since they address evolution at too low a level of abstraction (i.e., finding good hardware configurations at the FPGA configuration bit string level). This is analogous to trying to automatically generate complex software programs by searching the space of all possible bit strings, and it is unlikely that ibis approach will scale to complex tasks.

The only existing digital evolvable hardware work on a non-toy problem is the evolution of image compressi on algorithms and equalizers by Higuchi et al [t IMI+97]. However, their work req uires a custom-designed digital reconfigurable hardware platfor m in which each function block implements relatively bigtl-level arithmetic functions (in comparison with standard FPGAs, in which each function block implements logical functions. For digital evolvable hardware, it is desirable to usc commodity reconfigurable platforms developed by industry (i.e., FPGAs), in order to leverage industry-driven technology improvements (e. g., reconfiguration speeds, clock speeds, chip size, power consumption, etc.).

Wc propose an approach in which *algorithms* that solve the problems are automatically generated and adapted as a high-level programming language function (specifically, Lisp *s-expressions*). Wc wi II appl y the genetic programming [Koz92] technique (genetic algorithms applied to s-expressi ons) to search for s-expressio ns to achieve/optimize behaviors on the task. These algorithms (s-expressions) are then mapped onto a commodity reconfigurable hardware platform, e.g., a field-programmable gate ar ray (FPGA) using VI .SI CAD high-level synthesis/silicon compilation tools. Each of these components will be disc ussed in more detail below. A prototype system based on this approach was implemented for the task of lossless image compression.

A genetic programming system for evolving algorithms

Wc believe that bigll-level abstractions are essential in order for a searc h/optimi zation algorithm for evolvable hardware to find solutions to complex tasks. In general, the problem of combinator ial search/opt imizati on in nontrivial search spaces requires that the search algorithm is *biased* to search the regions of the scare. b space where good solutions are likely to exist (otherwise, the search algorithm wastes all of its time looking at regions where there are no good solutions). This search bias corresponds to human expert knowl edge which is applied to an otherwise intractable problem to make it tractable. To do ibis, we need to impose structure on the search space. This is not possible when the representation of the space that is being is searched is too low level. For example, if the representation of the space is the set of all possible FPGA configuration sirings, this space is much too low level for us to observe/i mpose any str ucture that would be of use when trying to automatically configure the FPGA to perform a image processing algorithm.

Thus, we propose the usc of a high-level, programming language as the representation over which a search algorithm scarL'tics for candidate soluti ons to a self-configuring hardware problem. In particular, we propose that we use Lisp s-expressions (based on the Lambda Calculus of Church) as the space which is being searched.

This enables us to apply techniques dc\'eloped in automatic programming, such as genetic programming [Koz92] to effectively search for algorithmic solutions to a given task. That is, we search the space of possible algorithms for a solu tion to a problem, rather than the space 01 all possible hardware configurations. The space of algorithms expressed as Lisp s-expression provides us with a search space which is significantly more amenable to search for good solutions. For example, wc can restrict the search space to the set of all *likely* solutions by restricting the set of functions and terminal sc(s that can be present in a candidate s-expression,

The search technique wc apply is genetic programming. Pioneered by Koza [Koz92], the techniq ue has been applied on a wide range of practical problems including robot control and image processing, While standard genetic algorithms [11oI75] typically apply biologically-inspired evolutionary operators to *fixed-length* representations Of task solutions, genetic programming applies analogous operators (selection, crossover, mutation) to tree-structured data str uctures such as LISP s-expressions. Figure 3. I shows an outline of an evol utionary algorithm. i n genetic programming, the individuals in the population are Lisp s-expressions. 1 .isp s-expressions are trees where the leaves arc terminals (either state data or constant values), and the internal nodes are functions.

```
t.=o
initialize P(t);
evaluate P(t) ;
while not. terminate do
    P'(t) := recombine P(t);
    P''(t) := mutate P' (t);
    evaluate P(t) ;
    P(t+1) := select (P''(t) i Q) ;
    t:=t+1;
end while
```

Figure:3. 1 Algorithm schema for an evolutionary algorithm P is a population of candidate solutions; Q is a special set of individuals that has to be considered for selection, e.g., Q=P(t).evaluate applies a domain-specific objective function to compute a objective function value for a candidate solution **(individual)'**

The end product of this component of the evolvable hardware system is a Lisp s-expression (see Figure 3.3 for sample s-expressions evolved for an image compression application).

A silicon compiler for mapping solutions to a standard FPGA

The other major component in the system is responsible for mapping of the s-expression which solves a problem to an actual configuration bit string which can be downloaded to a reconfigurable hardware platform such as an FPGA.

There is a significant body of recent work in the Electronic Design Automation (VLSI CAD) research community on *high-level synthesis* and *silicon compilation*, the problem of translating a programming language level behavioral specification of hardware to a hardware layout. We propose to leverage the existing synthesis/silicon compilation technology to enable the mapping from our tligll-level algorithmic solutions to an actual FPGA mapping. I ndeed, our proposal for the automatic generation of s-expressions which serve as the behavioral hardware specification for a CAD system can be considered a natural extension to the capabilities of silicon compilation technology, and has applicat ions for automa ted hardware design for application specific hardware systems (ASICs), as well as for self-cor~llgllt-it~g hardware systems.

The current state 0{ the art in silicon compilation" is such that "well-behaved programs", are easily mapped to an FPGA configuration. The system we propose to usc is HYPER [C PTR89] (developed at UC Berkeley), which takes as input expressions in the Silage dataflow programming language[Hil85]. The translation from the s-expressions generated by our genetic programming syste III to a Silage dataflow expression (which is then input to HYPER) is a very straightforward process and is easily automated. Currently, the translation is done by hand - we are now developing a s-expression to Silage translator.

The output of the silicon compiler can be downloaded directly to a standard FPGA such as the Xi linx FPGAs. Note that the compiler can be modified to target a range of hardware platforms, so our approach is not dependent on any particular platform, and is readily applied to new digital reconfigurable hardware plat forms which may become available in the future.

Optimization of hardware metrics

W c have presented a decomposition o f the self-configuing hardware system into 1) a genetic programming system which generates an s-expression (algorithm) to perform a given task, and 2) a silicon compilation system which takes the s-expression and maps it onto an FPGA config uration.

One potential concern about our approach is: since the genetic programming system is operating at the level of abstract algorithms, how can we optimize traditional hardware metrics in the resulting FPGA mapping such as hardware area usage, power consumption, and clock rate (speed)'! 1(is important to note the difference between *algorithmic metrics*, *e.g.,* the compression ratio in a image compassion algorithm, and *hardware implementation /11('trics* such as power consumption. The former are properties which are independent of the hardware implementation and can be easily measured by eval uating the algorithmic representation, while the latter can not be completely evaluated until the hardware mapping is available.

Since a complete, detailed mapping of an s-expression to an FPGA mapping is a time-consuming process ((he high-level synthesis can be done in about 0.5 minutes; detailed layout can take about 0.5 hours), we can not afford to actually perfo rm the full mapping to hardware for every candidate solution that needs to be evaluated by the genetic programming system (a typical run of a genetic programming requites that thousands of candidate solutions be evaluated).

Fortunately, there are algorithms (e.g., [RP91]) that can be usc(1 to estimate the hardware metrics of the resulting FPGA mapping from a bigtl-level algorithmic behavioral specification (i.e., the s-expression). Algorithms o f various fidelities exist, enabling a tradeoff between the speed at which the hardware metrics can be estimated (ranging from o. I seconds to up a full, actual place and route which takes 0.5 hours) and the accuracy O f estimation - the longer the time spent (m the estimation, the more accurate the results.

Thus, it is possible to add estimated hardware implementation metrics such as Speed and power consumption to the objective function used by the genetic programming system for the evolution of the s-expressions. By doing ibis, we can simultaneously optimize algorithmic and hardware implementation metrics.

A Prototype for Lossless Image Compression

To demonstrate the viabil ity of our approach, we developed a prototype genetic programming system to perform adaptive image compression based on a predictive coding compression algorithm. Image compression is an problem of' significant in space applications because the communications bandwidth between a spacecraft and ground is limited. An autonomous spacecraft sufficiently far from Earth that needs 10 send science data (images) needs 10 maximally compress the images given the limited communications bandwidth. Because image compression is extremely computationally intensive, a fast, hard ware implementation of a compression algorithm is desirable. A self-conllguring hardware system could be used to automatically generate a hardware-based image compression algorithm which is specially adapted for the class Of images captured by the spacecraft. Current Slate of the art lossless image compression algorithms include the CALIC algorithm of Wu and Memon [WM97] and the LOCO-I algorithm of Weinberger et al. [WSS96]. Reviews of' lossless image compression can be found in [MW97a,MW97b].

For an initial proof of concept, we considered the task of lossless image compression using a nonlinear predictive coding algorithm for which tbc nonlinear model was automati cally generated using a genetic program ming system. The resulting algorithm can be straightforwardly mapped to reconfigurable hardware, using a silicon compiler as described above, and demonstrates the feasibility of self-configuring hardware to image processing on an autonomous spacecraft.

Predictive coding is an image compression techniq ue which uses a compact model of an image to predict pixel val ues of an image based on the values of neighboring pixels. A *model* of an image is a function *model*(*x*,*y*), which computes (predicts) tbc pixel value at coordinate (*x*,*y*) of an image, given the values of some *neighbors* of pixel (*x*,*y*), where neighbors are pixels whose val ues are known. Typically, when processing an image in raster scan order (left to right, top to bottom), neighbors are se lected from the pixels above and to the left of the current pixel. For example, a common set of neighbors used for predictive coding is the set *{(x-1,y-1), (x,y-1), (x+1,y-1),(x-1,y)}*. *Linear predictive coding* is a simple, special case of predictive coding in which the model simply takes the average of the neighboring val ues. *Nonlinear* models assign arbitrarily complex functions to the 1I1OCIC1S.

Suppose that we ha ve a pet-kct model of an image, i.e., onc which can perfectly reconstruct an image given the pixel value of the border pixels (assuming we process the pixels in raster order). 'J'hen, tbc value of the border pixels and this compact model is all that needs to be transmitted in order 10 transmit the whole information content of the image. in general, it is not possible 10 generate a compact, perfect model of an image, and the model generates an *error signal* ((he differences at each pixel between the value predicted by the model and the actual value of the pixel in the original image.

There are two expected sources of compression in predictive coding based image compression (assuming that the predictive model is accurate enough). First, the error signal for each pixel should have a smaller magnitude than the corresponding pixel in the original image (therefore requiring fewer bits 10 transmit tbc error signal). Second, tbc error signal should have less entropy (ban the original message, since the model should remove of much of tbc ``principal components'' of the image signal[1]. To complete the compression, the error signal is compressed using a standard data compression technique such as Huffman coding or the dictionary-based Lempel-Ziv compression algorithms (c.f. [NG96]) as a ``back-en d'' compressor. Due to tbc two factors mentioned above, this should re sult in compressed data which is more compact than if the back-end compressor (e.g., Huffman coding) had been applied directly 10 the original image,

If we transmit this compressed error signal as well as the model, then a receiver can reconstruct the original image by applying an analogous decoding procedure (see Figure 4.2).

```
Encoder(Model, Image)
    for x= o to xmax
      for y=o to ymax
       error[x,y] . Image[x,y] - Model (x, Y)
Decoder(Model)
    f e z'x= 0 to xmax
      for y = O to ymax
        Image[x,y]= Model (x,y) + Error[x,y]
```

Figure: 3.2 Algorithm schema for predictive coding. Model(x,y) is a function that takes tbc coordinates of a pixel and returns a predicted value of that pixel. Image and Error are two-dimensional arrays

Given an image, our system uses genetic programming 10 generate a Lisp s-expression which is a nonlinear model for the predictive coding,

Since the s-expressions generated by the genetic programming system are intended to be mapped onto a F PGA, the set of functions and terminals was chosen carefully to enable efficient (with respect to space and speed) hardware m appings.

The terminals used for genetic programming were:

---

[1] If the model were perfect, then the error signals would consist of all 0's, and can be compressed to a single byte.

- values of the four neighboring pixels *Image[x-1,y-1], Image[x,y-1], Image[x+1,y-1], Image[x-1,y]*.
- constant values (generated randomly by the genetic programming algorithm).

The functions used were:

- arithmetic functions (+,-,*,/)[2]
- MIN(a,b) and MAX(a,b) functions which return the minimum and maximum values of (heir two argLlmc.ills, respectively.

Results / Discussion

The genetic programming system for evolving models for predictive coding image compression was evaluated by comparing the compression ratio[3] using the resulting models against standard lossless compression techniques on a set of gray scale images. "

For purposes of comparing the compression provided by our system, we must consider the total size of the data which must be transmitted to a receiver in order to allow the lossless reconstruction of the original image.

A standard back-end compression algorithm needs to be applied to the error image. For this experiment, we used the Unix *compress* utility, which appl ics adaptive Lempel-Ziv encoding to a file.[5] In addition, note that given the four pixel ncigbhot-hood we use, the pixel values of the borders of the image, i.e., the top row, the leftmost col umn, and the rightmost column need to be stored directly (these are the border cases for which we can not apply the predictive model). Also, the model (which is uniq ue for each image) must also be stored in the compressed image data. We applied Unix *compress* to the border pixels and the model, and concatenated these to the compressed error signal. Finally, two integer values indicating the size of the image (height, width) were added to the file. Given this data, we can reconstruct an image without loss of information.[6]

Thus, the size of the compressed image is: *sizeof(CompressedError) + sizeof(CompressedBorder) + sizeof(CompressedModel) + sizeof(2 integers)*

The following images were used for evaluation:

- *dsn-tall*: an image of' fault lines and clusters of volcanic domes, ranging from I.5 to 7.5 km in diameter, on the surface of Venus.
- earth-vicar: a satellite image of the earth

For each image, the genetic programming system was run once, with a population of 2000 and the n umber of generations set to 10. Other parameters (e.g., were set to standard val ues as described in [Koz92]).

The compression ratio of the following arc shown in '1'able 3.1

- *evolved*: The evolved predictive coding compression algorithm
- *CALIC*: A state of the art lossless image compression algorithm, described in [WM97].
- *GIF*: GIF compression (a standard lossless image compression technique).
- *JPEGLS*: The recently established lossless JPEG standard (formerly known as 1 .OCO-I)

Image	*raw size*	*JPEGLS*	*CALIC*	***Adaptive***
earth-vicar	87591	52891	50763	**38171**
dsn-tall	783232	22362.4	171537	**144338**

Table 3.1 Compression algorithm performance on two science images (file size in bytes).

Figure 3.3 shows a sample nonlinear model (s-expression) evolved by the genetic programming system for a test image.

As '1'able 3.1 shows, the compression ratio using the evolved models is, superior to the other methods for the test images used in the experiments. Note that the evolved models yield a significant improvement over CAI IC, which is currently the best known lossless image compress ion algorithm. Furtherl more, these results were obtained without any special tuning 01 algorithm control parameters for the genetic programming (it is well known that to maximize performance for a particular problem, the control parameters for genetic programming such as population size, crossover rate, etc. need to be tuned for the particular problem domai n).

However, it should be noted that the genetic programming system takes several orders of magn itude more lime to evolve a model that achieves its superior results (several hours per image) than the other approaches (which run in a few seconds).

In addition, the performance of a compression algorithm depends largely on the class of image to which it is

---

[2] The division operator used was special *protected division* operator which returns 0 in case of division by zero. This is common practice in genetic programming, in order to prevent arithmetic exceptions.

[3] The *compression ratio* of compressed data is the size of the compressed data divided by the size of the original data.

[4] In this report, we focus on grey scale images, but the technique can be straightforwardly extended to color images by operating on three image planes (red, blue, green).

[5] In future work, we will investigate other back-end compressors such as arithmetic coding.

[6] We implemented a decompression program that reads a compressed image file and reconstructs the original image.

applied. Although our results are very promising for the set of test images we used, a wider range of test images needs to be used. We are currently trying to obtain the set of benchmark images used for the "Next Generation Lossless Compression of Continuous-tone Still Pictures" effort by the ISO (ISO/IEC JTC 1 N2395) and evaluate our approach using this standard image set.

In practice, the evolvable hardware approach is likely to be most useful in a context in which a large number of images from the same class need to be compressed and transmitted. As an example, we envison the following scenario: consider a deep space probe which needs to send thousands of similar images (e.g., atmospheric images) from the mission target (say, Pluto) back to Earth. The probe would first sample a few images and evolve a single nonlinear predictive model which is likely to perform well for the set of images that need to be transmitted. This could be achieved, for example, by a simple modification to our system in which the evolving models are scored according to their average performance on a small subset of the images, rather than on a single image. This model would then be mapped onto the on-board FPGA using the silicon compiler, and the entire set of images would be compressed using this model and transmitted back to Earth. An alternate strategy would be to download a few sample images back to Earth, where nonlinear model is evolved for a sample of the images using a high-performance ground-based computer,[7] and is uploaded back to the spacecraft. This uploaded model is then mapped on the on-board FPGA and used to compress all the images, which are then sent back to Earth. We will investigate this scenario further in future work.

```
(- (+ (- X4 0.49115) -0.6'/ 691)
   (/  (+  (-  (+ (+ X2
                  (- X4
                     (- xl  xl)))  X4)  xl)  X4)
      (+ (+ (/ -0.06982 -0.18'/ 61)
            (+X4 -0.46071))
         (/ x 4
            (+ -0.7318"/ x4)))))
```

Figure: 3.3 Evolved s-expression for a nonlinear predictive model of the face image

4. CONCLUSIONS/FUTURE WORK

In this paper, we have described the following:

- Development of an architecture for digital evolvable hardware based on a genetic programming system that generates algorithmic representations of solutions to problems and a silicon compiler which maps the algorithm to hardware (a commodity FPGA).
- Proof of concept implementation and evaluation of a software prototype of the genetic programming component of an evolvable hardware system for lossless image compression. The results are quite promising (compression ratios better than state of the art lossless algorithms were achieved).

We intend to extend this work in several directions. We will implement the translator from Lisp s-expressions to Silage dataflow expressions, which will enable us to use the HYPER silicon compiler and provide us with a complete, digital evolvable hardware system framework.

A particularly interesting research topic is the optimization of hardware implementation. Exploration of this topic is likely to yield results that are applicable not only to evolvable hardware for space applications, but for electronic design automation (VLSICAD) in general.

The work on the prototype lossless image compression can be extended in a number ways. Additional function/terminal sets can be used for the genetic programming system. Of particular interest is the discovery of a set of minimal functions which provide good compression and are particularly amenable to FPGA mapping. With respect to alternative terminal sets, this work will be focused on the use of different neighborhoods (recall that we used a simple 4 pixel neighborhood for predictive coding). We currently apply a single model to the entire image, one way to obtain better compression is to split the images into blocks (e. g., 32 pixel by 32 pixel blocks) and evolve a separate model for each block. We note that by quantizing the error signal, it is possible to perform lossy image compression, a related problem with many practical applications.

Finally, we will extend the scope of our investigation by applying our architecture to other image processing algorithms such as edge detection

In additoin to digital evolvable hardware, we are currently developing analog digital hardware. "The-c is currently no standard reconfigLlt'able analog platform (i.e., the equivalent of an FPGA in the analog domain). Thus, we are developing candidate reconfigurable analog hardware platforms which can be used as the foundation for analog evolvable hardware.

Rapidly Reconfigurable Analog Computer for Evolving Dynamic Systems

Another direction that we started exploring in evolvable hardware is related to the\ 'ol\"itlgco lllI>lcxcly rl:ltllic systems (with applications for example to spacecraft control" problems). We have developed an algorithm which makes use of a rapidly reconfigurable analog computer

[7] Because genetic algorithms are extremely parallelizable, it should be possible to perform the evolution of the model in a few minutes on the ground.

implementing dynamic systems. A Sla[c-space decomposition of the dynamic system is made, and then local optimization, (hill climbing) is used to obtain the desired behavior (i.e., the control problem sol ution). We have designed and implemented a board level prototype for the above hardware architecture. The current bead level prototype can switch among 2048 configurations (1 ills/col]tigLtrilli or~)." Wc have demonstrated the use of this architecture for a functi on approximation problem. In addition to evol vable hardware, this hardware can be used in a large variety of' applications, e.g. as analog math co-processor for physical science simulations, to perform Cellular Neural Networks style fast image processing, etc.

We arc extending this work as follows: wc are extending the above simulation work, beyond the function approximation stage, to evolve complete dynamic systems using the above slate-spare decomposition method. Wc will test the above technique by evolving control problem benchmarks.

Architecture Design and Simulation of a CMOS Chip for Analog Evolvable Hardware

Koza et al. [Koz96]have evolved in simulation a variety of analog circuits. Unfortun ately, their approach can not be used directly for evolving in hardware. Also, currently, there is no reconfigurable analog hardware suitable for evolution (i.e. with enough resources and fast reconfiguration). Wc are investigating the design and evolution of an *evolvable CMOS chip.*

Wc have performed a series of designs and simulations. When the search space was bounded by the number of components a priori known as the minimal number of components providing the desired functionality, evolution in the space of electronic components proved hind. For known circuit topologies wc evolved the values for each PMOS/NMOS transistor's channel Width and Length. Wc are currently devel oping an ar chitecture that would support such implementation and wc will attempt a simulated evolution on the designed evolvable CMOS chip. This approach will enable evolving analog circ uits directly in hardware.

ACKNOWLEDGEMENTS

The research described in this paper was performed by the Center for Space Microelectronics Technology, Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the JPL Director's Discretionary Fund.

REFERENCES

[CPTR89] C.M. Chu, M. Potkonjak, M. Thaler, and J. Rabaey. Hyper: an interactive synthesis environment for high performance real time applications. In *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pages 432-5, 1989

[}1?1S97] }1. Hemmi, T. Hikage, and K. Shimohara. Adam: A hardware evolutionary system. In *IEEE International Conference on Evolutionary Computation*, pages *193-196,* 1997.

[Hil85] P. Hilfinger. A high-level language and silicon compiler for digital signal processing. In *Proceedings of the IEEE 198.5 Custom Integrated Circuits Con ference*, pages 213-10, 1985.

[HMI+97] T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, W. Liu, and M. Salami. Evolvable hardware at function level. In *IEEE International Conference on Evolutionary Computation*, pages 187-192, 1997.

[Hol75] J. Holland. *Adaptation in natural and Artificial Systems.* University of Michigan Press, 1975.

[JCC+92] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang. Evolution as a theme in artificial life: The genesys/tracker system. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life //, pages* 549-577. Addison-Wesley, 1992.

[KBAK96] J. Koza, F.H. Bennett, D. Andre, and M.A Keane. Automated wywiwyg design of both the topology and component values of analog electrical circuits using genetic programming. In *Proceedings of Genetic Programming Conference*, pages 28-31, 1996.

[Koz92] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[MW97] N. Memon and X. Wu. Recent progress in losslessimagecoding.*The Computer Journal, to appear*, 1997.

[Nit?'ar] N. Memon and X. Wu. Lossless compression. In *CRC Handbook of Communication.* 1996 (to appear).

[NG96] M. Nelson and J-t . . Gailly. *The Data Compression Book (second edition).* M&T Books, 1996.

[RP91] J. Rabaey and M. Potkonjak. Complexity estimation for real time application specific circuits. In *ESSCIRC '91. Seventeenth European Solid State Circuits Conference. Proceedings*, pages 201-204, 1991.

[Tho96a] A. Thompson. An evolved circuit, intrinsic in silicon, entwined in physics. In *International Conference on Evolvable Systems.* Springer Verlag Lecture Notes in Computer Science, 1996.

[Tho96b] A. Thompson. Silicon evolution. In *Proceedings of Genetic Programming Conference*, 1996.

[WM97] X. Wu and N. Memon. Context-based, adaptive, 10 SSICSS image codes. *IEEE Transactions on Communications*, *45(4)*, 1997.

[WSS96] M.J. Weinberger, G. Seroussi, and G. Sapiro. Loco-i: A low complexity, context-based, lossless image compression algorithm. In *Proceedings of the Data Compression Conference (DCC'96)*, pages 140-149, 1996.